Toward a Dynamic Programming Solution for the 4-peg Tower of Hanoi Problem with Configurations

Neng-Fa Zhou and Jonathan Fruhman

Department of Computer and Information Science, Brooklyn College
The City University of New York
New York, NY 11210-2889, USA

Abstract. The Frame-Stewart algorithm for the 4-peg variant of the Tower of Hanoi, introduced in 1941, partitions disks into intermediate towers before moving the remaining disks to their destination. Algorithms that partition the disks have not been proven to be optimal, although they have been verified for up to 30 disks. This paper presents a dynamic programming approach to this algorithm, using tabling in B-Prolog. This study uses a variation of the problem, involving configurations of disks, in order to contrast the tabling approach with the approaches utilized by other solvers. A comparison of different partitioning locations for the Frame-Stewart algorithm indicates that, although certain partitions are optimal for the classic problem, they need to be modified for certain configurations, and that random configurations might require an entirely new algorithm.

1 Introduction

The classic 4-peg Tower of Hanoi problem is posed as follows:

Given: Four pegs, and n disks of differing sizes stacked in increasing size order on one of the pegs, with the smallest disk on top.

Goal: Stack the n disks on a different one of the four pegs, using the following rules:

1. Only one disk can be moved at a time.
2. Only the top disk on any peg can be moved.
3. Larger disks cannot be stacked above smaller disks [15].

A number of authors have created algorithms to try to solve both this puzzle and the related puzzle involving an arbitrary number of k pegs. Frame and Stewart provided the first documented algorithms for the k-peg problem [4] [15]. Their algorithms involved partitioning the disks into intermediate sub-towers. Since the time that Frame and Stewart published their results, others have analyzed algorithms in order to either attempt to solve the 4-peg and k-peg Tower of Hanoi puzzles using a minimal number of moves, or to try to prove the optimality of the number of moves generated by Frame's and Stewart's algorithms. Section 2 discusses the Frame-Stewart algorithm.

The goal of this paper is to make steps toward an efficient dynamic programming solution for the 4-peg tower of Hanoi puzzle. The solution is presented using the B-Prolog programming language [24], and uses tabling, a technique similar to pattern databases [7], in order to decrease the number of necessary computations. The program will be *non-deterministic*, in order to extend the Frame-Stewart algorithm to variants of the 4-peg problem that involve disk configurations. In order to reduce the non-determinism, this paper examines different partition locations for the intermediate sub-towers, in order to find the most efficient one to use for the configuration problem. Although there is no guarantee that the 4-peg Frame-Stewart algorithm is optimal [10], the dynamic programming solution utilizes this algorithm, as it has been proven to generate the optimal solution for up to 30 disks [7].

In Sect. 3, the configuration problem is examined, as posed by the 2011 ASP Competition [19]. The focus is team BPSolver's program [28], which uses Prolog together with tabling [27]. BPSolver's results are compared with those of the other teams: team Fast Downward, which used PDDL, and teams EZCSP, IDP, Potassco, and Aclasp, which used SAT solvers and grounders[1]. The comparison will concentrate on Potassco's program, which had the second-best performance for the Hanoi benchmarks. Sect. 3.3 compares dynamic programming approaches for splitting the problem into sub-problems, finding one that seems optimal.

Section 4.1 studies the seemingly optimal solution when applied to random configurations that do not extend from Frame's and Stewart's algorithms. Then, in Sect. 4.2, the dynamic programming approaches are applied to the classic problem for up to 30 disks, getting a different result than Sect. 3.3. Finally, Sect. 5 describes alternative approaches to solving the 4-peg and k-peg problems.

2   The Frame-Stewart Algorithm

When the k-peg Hanoi problem was posed in 1941, two authors provided solutions. One algorithm, provided by J. S. Frame, is iterative. The other, written by B. M. Stewart, the proposer, is recursive. Both authors claim that their algorithms generate the minimum number of moves, while Frame admits that other methods might also be able to provide the same minimum number.

Following is Stewart's algorithm for the 4-peg problem:

1. Move the Mid topmost disks to another, intermediate, peg (which is not the destination peg), using all 4 pegs.
2. Move the $n - Mid$ remaining disks to the destination peg using the 3 remaining pegs.
3. Move the Mid disks from their current peg to the destination peg, using all 4 pegs [15].

Frame's algorithm is similar to Stewart's. According to Frame, for the k-peg problem, a series of towers needs to be created. The smallest tower will consist

---
[1] https://www.mat.unical.it/aspcomp2011/Participants/

of the largest disks, and the largest tower will consist of the smallest disks. Only the largest disk remains on the start peg. Once the towers are created, the largest disk is moved directly from the start peg to the destination peg. Then, the tower of the next-largest disks can be moved to the destination peg using three pegs. Each intermediate tower is then moved to the destination peg using one more empty peg than the previous tower [4].

If the goal is to minimize the number of disk movements, then this algorithm presents two problems. One problem is to find the optimal number of disks, Mid, to place on the intermediate peg. This paper examines the partitioning issue in Sect. 3.3, using Stewart's recursive algorithm. The other issue is that Frame and Stewart do not provide evidence that an optimal aglorithm involves the creation of intermediate sub-towers [10]. However, as mentioned above, since Korf has verified the optimality of this algorithm for 4 pegs and up to 30 disks [7], the Frame-Stewart algorithm is a good estimate.

3 The ASP Competition Problem

3.1 The 4-Peg Problem with Configurations

One of the ASP Competition's problems modifies the 4-peg puzzle to include configurations. This problem follows the same rules as the classic problem, but differs in regard to how the puzzle is initially arranged, and regarding the goal of the puzzle. Instead of all the disks beginning and ending on the same peg, this problem provides two *configurations*. The first configuration is the *start state*, a distribution of the disks over any combination of all four pegs. The second configuration is the *goal state*, a different distribution of the disks over the four pegs. These configurations were created based on the moves that the Frame-Stewart algorithm would perform. The problem is to generate the disk moves needed to get the disks from the first configuration to the second. In addition, there is a bound on the number of moves that could be generated [19].

The BPSolver team used B-Prolog to generate solutions for the given configurations. The following is the relevant code:

```
:-table plan4(+,+,+,-,min).
plan4(N,_CState,_GState,Plan,Len):-N=:=0,!,Plan=[],Len=0.
plan4(N,CState,GState,Plan,Len):-
    remove_largest_disk_if_in_place(
            N,CState,GState,CState1,GState1),!,
    N1 is N-1,
    plan4(N1,CState1,GState1,Plan,Len).
plan4(N,CState,GState,Plan,Len):-
    % split disks into two groups
    partition_disks(N,CState,GState,ItState,Mid,Peg),
    % sub-problem1
    remove_larger_disks(CState,Mid,CState1),
    plan4(Mid,CState1,ItState,Plan1,Len1),
```

```
% sub-problem2
remove_smaller_or_equal_disks(CState,Mid,CState2),
remove_smaller_or_equal_disks(GState,Mid,GState2),
N1 is N-Mid,
plan3(N1,CState2,GState2,Peg,Plan2,Len2),
% sub-problem3
remove_larger_disks(GState,Mid,GState1),
plan4(Mid,ItState,GState1,Plan3,Len3),
%
append(Plan1,Plan2,Plan3,Plan),
Len is Len1+Len2+Len3.
```

Instead of focusing on the given upper-bound of moves, this program attempts to find the *smallest* number of moves required to get from the start configuration to the goal configuration. Each configuration is represented by a single state, consisting of four lists. Each list represents a single peg, and stores the disks that are currently located on the peg. This is an improvement on the BPSolver team's prior attempts to represent configurations using Boolean expressions. Prolog lends itself to list operations, and the number of disks on a peg is simply the length of the list.

The predicate `plan4` is defined with three clauses. The first clause is the termination condition. When the number of disks to be removed is zero, the problem has been solved. The second clause determines whether the current largest disk is currently in place. If so, it simplifies the problem by logically removing the disk.

The third clause uses a modified form of Stewart's approach to solve the configuration problem. The disks are split into two sub-groups, one of which is placed in an intermediate tower. The problem is separated into three sub-problems:

1. The smallest `Mid` disks are placed on an intermediate peg, `Peg`, by calling `plan4` recursively.
2. The larger disks are moved from their current pegs to their destination positions by using the deterministic 3-peg algorithm on all of the pegs except for `Peg`[2].
3. The small disks in the sub-tower are moved from the intermediate peg to their destination positions by calling the `plan4` predicate recursively [27].

Another important line of code is `:-table plan4(+,+,+,-,min)`. This uses *tabling* to store information about each state in memory. The purpose of tabling is to store answers to sub-goals, and to utilize the answers for future variant sub-goals. This is a useful tool for dynamic programming, which reuses solutions to overlapping sub-problems. B-Prolog uses linear tabling, lets variant sub-goals share answers, and uses the local, or lazy, strategy to return answers [25]. The

---

[2] The 3-peg Tower of Hanoi is deterministic, as it has been proven to have a minimum solution of $2^n - 1$ moves for n disks [1].

most recent version of B-Prolog replies on hash-consing to let tabled subgoals and answers share ground structed terms [26].

There are two benefits to using tabling. The first benefit is that tabling prevents infinite loops. Once a state is visited, it should not be revisited. If states are visited more than once, the program could be stuck cycling between multiple states, possibly by just repeatedly moving a single disk between two pegs. By storing states in memory, the program can check the table to see if a state has already been encountered. The other benefit of using tabling is that it reduces the number of calculations. Once the program knows how to move p disks between two pegs, it can check the table to determine how to move any other set of p disks between any two pegs. Sub-problems are represented in such a way that the same problem has the same representation and can share answers through tabling. It does not matter what the sizes of the p disks are, nor does it matter which pegs are being used as the current start and destination pegs, assuming the intermediate and destination pegs are logically empty, meaning that they do not contain disks smaller than the largest one being moved. This decreases the number of operations used during the recursive calls, and is helpful when backtracking to test a different solution.

The line :-table plan4(+,+,+,-,min) goes together with the line defining the predicate plan4(N,CState,GState,Plan,Len). A plus-sign (+) indicates that the corresponding arguments (N - the number of disks, CState - the current state, and GState - the goal state) are input. A minus-sign (-) indicates that the corresponding arguments (Plan - the sequence of disk moves) are output. The last part, min indicates that Len, the length of the plan of disk moves, should be minimized. By minimizing the plan length, the number of moves will clearly be within the bounds provided in the given problems; otherwise, the problems would be unsolvable.

Since the predicate plan4 is non-deterministic, it presents a few interesting issues. Like the classic algorithm, this program must determine optimal sizes of the disk sub-towers. A new problem that arises is that the program must determine which peg to use to store each sub-tower. In the classic 4-peg problem, there is one peg that never has to be used to store a sub-tower [9] [20]. However, the same is not true for the configuration problem. The start and destination pegs can change for each disk, meaning that any peg might need to be used to store a sub-tower. For further discussion of these issues, see Sect. 3.3.

Tabling is also used for plan3, defined below for the 3-peg problem.

```
:-table plan3(+,+,+,+,-,min).
plan3(0,_CState,_GState,_UnusedPeg,Plan,Len):-!,Plan=[],Len=0.
plan3(N,CState,GState,UnusedPeg,Plan,Len):-
   remove_largest_disk_if_in_place(N,CState,
                          GState,CState1,GState1),!,
   N1 is N-1,
   plan3(N1,CState1,GState1,UnusedPeg,Plan,Len).
plan3(1,CState,GState,_UnusedPeg,Plan,Len):-!,
   Plan=[(Peg1,Peg2)],Len=1,
```

```
        btm_disk_on_peg(1,CState,Peg1),
        btm_disk_on_peg(1,GState,Peg2).
  plan3(N,CState,GState,UnusedPeg,Plan,Len):-
        btm_disk_on_peg(N,CState,Peg1),
        btm_disk_on_peg(N,GState,Peg2),
        other_two_pegs(Peg1,Peg2,Peg3,Peg4),
        (UnusedPeg==Peg3->TmpPeg=Peg4;TmpPeg=Peg3),
        N1 is N-1,
        remove_bottom_disk(CState,Peg1,CState1),
        ItState=s(_,_,_,_),
        Tower @= [I : I in N1..(-1)..1],
        foreach(I in 1..4,
                (I==TmpPeg->arg(I,ItState,Tower);
                 arg(I,ItState,[])
                )
        ),
        plan3(N1,CState1,ItState,UnusedPeg,Plan1,Len1),
        remove_bottom_disk(GState,Peg2,GState1),
        plan3(N1,ItState,GState1,UnusedPeg,Plan2,Len2),
        append(Plan1,[(Peg1,Peg2)|Plan2],Plan),
        Len is Len1+Len2+1.
```

Like `plan4`, the predicate `plan3` has a clause for the termination condition and a clause for reducing the problem when the latest disk is in place. When the number of disks to be moved is one, it takes one step to solve it. Otherwise, the problem is divided into three sub-tasks: the first sub-task is to move N-1 disks except for the largest one from the current peg to a temporary peg; the second sub-task is to move the largest disk to the destination peg; the third task is to move the N-1 disks from the temporary peg to the destination peg. Unlike `plan4`, `plan3` is deterministic.

3.2   Competition Results

The ASP Competition had six participants, including BPSolver[3]. FastDownward used Planning Domain Definition Language to represent the problems, and utilized A* search with the selective-max and landmark-cut heuristics, using the input to bound the maximum solution length, in order to solve the Hanoi problem. The IDP team described the problems by using First Order Logic with Inductive Definitions, translated the problems into Extended Conjunctive Normal Form by using the Gidl grounder, and solved the problems with the MINISAT(ID) solver. The remaining three teams solved the Hanoi problem

---

[3] The following implementation details, and the participants' programs, can be obtained at the team description pages, located at https://www.mat.unical.it/aspcomp2011/Participants/. Further details can be found at the solvers' websites, as listed in the teams' descriptions online.

by using the grounder Gringo, which translates input programs into equivalent, variable-free programs, and the solver Clasp, which focuses on answer set programming together with nogood learning, checking for violated constraints. While EZCSP ran Clasp on its default settings, Aclasp utilized a modified restart strategy, which, as described by the team, "depends on the average decision-level on which conflicts occurred." In order to solve the Hanoi problem, the final team, Potassco, specified that Clasp should use the Variable State Independent Decaying Sum decision heuristic, and limited the amount of preprocessing and the initial database size.

The ASP Competition programs were graded based on two criteria: the correctness of the solution and the amount of time that it took for each program to find the solution. The competition organizers had sixty instances[4] that could have been used to test the Hanoi solvers. Only fifteen of those instances were actually used to grade the solvers. Based on those fifteen instances, the BPSolver team scored highest, followed by Potassco, AClasp, and IDP. EZCSP and Fast Downward tied for the lowest score.

BPSolver's program was the only one that actively used the Frame-Stewart algorithm to limit the size of the search space. Every other program defined legal moves for the tower of Hanoi problem, and used the definition as input to their solvers. Therefore, the other teams' scores were a result of the solvers they used and the heuristics that the solvers employed to trim the search space. The active use of the Frame-Stewart algorithm contributed to the speed of BPSolver's program. Although some of the other teams used heuristics, and added rules in order to improve the search, such as EZCSP specifying that no disk should be moved twice in a row, they still had a larger search space. As will be shown in Sect. 3.3, BPSolver used a mathematical function to partition the disks, which limited the possible moves at any given time. In addition, tabling reduced the number of repeated calculations, causing BPSolver's program to run faster than all the other programs.

In the competition, BPSolver's Hanoi program scored 94, while Potassco's scored 81. However, an analysis of their performances on all 60 instances, summarized in Table 1, shows that the margin would not be as large. BPSolver still scores higher than Potassco, but that is primarily due to the speed of B-Prolog, as opposed to the correctness of the program.

Table 1. Comparing B-Prolog's original program with Potassco's program

Sys	AVG Seconds	MAX Seconds	Num Solved	SSolve	STime	Score
BPSolver	0.022883333	0.047	54	45	45	90
Potassco	10.61766667	102.867	60	50	37	87

---

[4] http://www.mat.unical.it/aspcomp2011/files/HanoiTower/hanoi_tower-full_package.zip

BPSolver's program printed a result of "UNKNOWN" for 6 of the instances, while Potassco's was able to solve all instances. Four of those six instances are a result of an error[5], in which the program printed "UNKNOWN" after finding a solution. Only one of these instances was run during the competition. The remaining two unsolved instances, which are illustrated in Fig. 1, and which were not run during the competition, are indicative of a larger problem. Team BP-Solver mistakenly believed that there was a known optimal guide for partitioning the disks when creating intermediate sub-towers [27]. The B-Prolog program's partitioning guide was not always able to find an optimal solution.
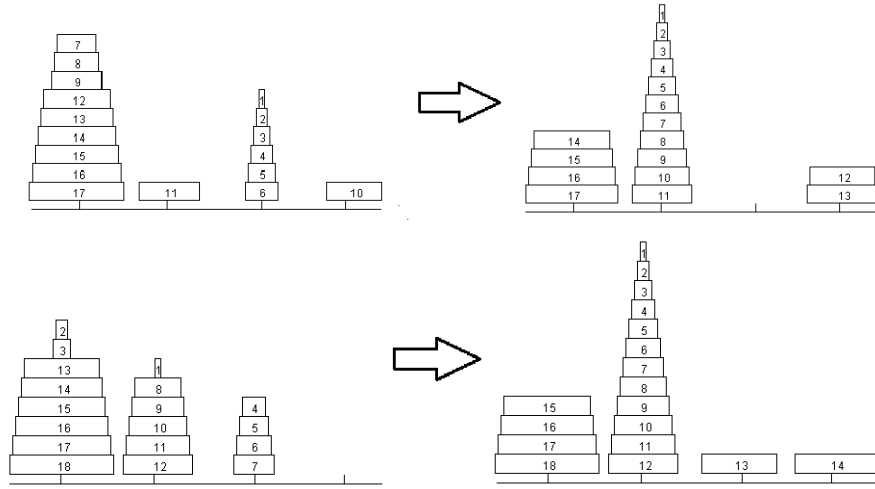

Fig. 1. Instances 17 and 22, the unsolved instances

3.3 Partitions

There have been a number of different estimates for determining the optimal size of the intermediate sub-tower when given n disks.

For the ASP competition, BPSolver used the following code to determine the optimal partition location:

```
partition_disks(N,CState,GState,ItState,Mid,Peg) :-
    btm_disk_on_peg(N,CState,Peg1),
    btm_disk_on_peg(N,GState,Peg2),
    other_two_pegs(Peg1,Peg2,Peg3,Peg4),
    (Peg=Peg3;Peg=Peg4),
```

[5] When a sub-tower is built, it should be built on a logically empty peg. The original program did not test if the peg was empty, causing the error.

```
arg(Peg,CState,CDisks),
arg(Peg,GState,GDisks),
PN is N-integer(sqrt(2*N)+0.5),
Low is max(PN-2,1),
Up is min(PN+2,N-1),
between(Low,Up,Mid),
(CDisks=[CBDisk|_],CBDisk=<Mid; CDisks=[]),
(GDisks=[GBDisk|_],GBDisk=<Mid; GDisks=[]),
Tower @= [I : I in Mid..(-1)..1],
ItState=s(_,_,_,_),
foreach(I in 1..4,
        (I==Peg->arg(I,ItState,Tower);
         arg(I,ItState,[])
        )
).
```

The arguments are similar to the ones in the `plan4` predicate, except `ItState` defines the intermediate state with the sub-tower, and `Peg` is the peg on which the sub-tower will be placed.

As mentioned above, there is an additional issue about where to place the sub-tower, based on the configurations. The program solves it in one of two ways. If the current configuration has a pre-existing sub-tower, a new one does not need to be created. Otherwise, the program non-deterministically tests both of the pegs that are currently serving as intermediate pegs for the optimal sub-tower location.

Given n disks, the program uses Rand's estimate of $n - \lfloor \sqrt{2n} + 0.5 \rfloor$ for the sub-tower size. As discovered after the ASP Competition, this does not always generate the optimal solution. Therefore, the program was tested using different partition estimates. Table 2 shows the results[6].

Table 2. Partitions

Partition	Source	AVG Seconds	AVG Table Used (Bytes)	MAX Seconds	MAX Table Used (Bytes)	UNKNOWN
$N - \lfloor \sqrt{2*N} + 0.5 \rfloor$	[12]	0.0234	121135.4	0.047	207724	2
$N - \lceil \sqrt{2*N} + 0.25 - 0.5 \rceil$	[20]	0.0209	121135.4	0.032	207724	2
$N - \lfloor \frac{\sqrt{8*N+1}-1}{2} \rfloor$	[1] [13] [22]	0.023	141369.4	0.032	206728	0
$N - K$, where K is the smallest integer such that $T_K \geq N$ (with tabling of results).	[23]	0.021566667	122005.1333	0.032	208868	2
$N - K$, where K is the largest integer such that $T_K \leq N$ (with tabling of results).	[1]	0.02285	142401.9333	0.032	207960	0
$N - K$, if $N = T_K$ for some K Otherwise, N - K, or N - (K + 1), where K is the largest integer such that $T_K < N$ (with tabling of results).	[17]	0.025833333	162988.3333	0.047	237360	0
Program Decides Without a Guide		0.039466667	4457510.067	0.124	16884172	0

---

[6] All of the tests described in this paper used B-Prolog version 7.5, which is the version used for the competition. Later versions of B-Prolog perform tabling in a different manner, decreasing the amounts of used table memory in some cases, and increasing the amounts used in other cases.

For Table 2, two types of estimates were used. One type is a formula stated in terms of the number of disks. The other type relates the 4-peg Hanoi problem to the *triangular numbers*, which are of the form $T_k = \frac{k*(k+1)}{2}$ [13], by finding the triangular number that is closest to the number of disks. The estimates of the first type explicitly state the relation between the number of disks and the triangular numbers found by the second type [1]. In the final row of Table 2, the program was not given a guide for where to partition the disks. Instead, it tested every number between 1 and $n-1$, where n is the number of disks.

There are a number of other guides for partitioning the disks [6] [9] [11] [17], but these were not tested. Some, such as [9], are too closely related to specific algorithms. Others, like [6], are too mathematically complex to test in terms of this program. Some of the remaining, like [11], are for specific cases. Some alternative partitions are clearly incorrect, as shown by Stockmeyer [17]. Instead of including Frame's and Stewart's estimates, this study incorpates Stockmeyer's estimate, which is based on those of Frame and Stewart [17].

The B-Prolog program allows for an error in the number, PN, returned by the estimates. It checks all values between $PN - 2$ and $PN + 2$ for an optimal partition number. If this error bound is removed, and the program uses the exact number that is found, or if the error bound is decreased to $PN - 1$ and $PN + 1$, then none of the functions in Table 2 solve all of the instances. It should be noted that three of the estimates in Table 2 fail for two of the instances, 17 and 22, even when the error is allowed.

The different approximations were tested for time and the amount of table memory that they used. On average, the explicit functions almost always used less memory than the partitions that calculated each triangular number until a certain condition was met. The maximum memory used was always smaller for the explicit functions than for the repeated calculations. One reason for this is that tabling was incorporated into the calculations of the triangular numbers in order to reduce the number of required calculations. As shown in Table 2, if the program tests every possible partition number, then it is guaranteed to find the optimal solution. However, the time and memory costs are much higher than they are if the program is given a partitioning guide.

Table 2 indicates that the best guide to use for the competition is the one originally provided by Rohl and Gedeon (as shown in row 3). It should be noted that, if floor and ceiling operations are removed, Liefvoort's solution (row 2) is mathematically equivalent to theirs. Since Rohl and Gedeon use the floor operation, while Liefvoort uses the ceiling operation, their solutions differ.

After the original program's error was removed, and the partitioning estimate was changed to that of Rohl and Gedeon, BPSolver's modified program was run on all 60 instances. Table 3 shows the results of these tests, as compared to Potassco's results. The results show that Rohl's and Gedeon's numbers are better than Rand's, because BPSolver now scores 100, instead of 94.

Table 3. Comparing B-Prolog's modified program with Potassco's program

Sys	AVG Seconds	MAX Seconds	Num Solved	SSolve	STime	Score
BPSolver	0.023	0.032	60	50	50	100
Potassco	10.61766667	102.867	60	50	37	87

4 Random Configurations and the Classic Problem

Each of the sixty instances generated for the 2011 ASP Competition consisted of configurations that would be created during the execution of the Frame-Stewart algorithm on the classic 4-peg problem. The configurations included pre-created sub-towers that the Frame-Stewart algorithm would have generated. This section will show that, although a certain partition number is shown to work on the ASP configurations, it is not guaranteed to solve every possible random configuration.

BPSolver's modified program was tested in two additional ways. It was run on randomly generated configurations, some of which might not be created during the execution of the Frame-Stewart algorithm, and it was run on the classic 4-peg problem.

4.1 Random Configurations

The first test was running the BPSolver program on randomly generated configurations. They were generated to match the ASP Competition's input files, which consisted of five sets of predicates [19]. The BPSolver program was tested on twenty random configurations. Since the configurations were random, some appeared to contain pre-existing intermediate sub-towers, while others did not. Two versions of BPSolver's program were tested. One version created the sub-towers using the formula originally posed by Rohl and Gedeon [13] with an error bound of two, and the other did not use any guide for where to partition the disks.

These tests had varied results. Only twelve of the twenty instances were solved by both programs. These instances had between 18 and 20 disks, while their solutions required between 120 and 257 steps. Another five instances were only solved by the program that did not use a partitioning guide. These instances had between 21 and 23 disks, and required between 387 and 408 steps. One instance, with 25 disks, was solved in 533 steps by the program that used a formula, while the program that did not use a guide ran out of memory. The remaining two instances, which had 24 and 25 disks, were solved by neither program.

The results appear to indicate that the 4-peg problem with random configurations requires a different approach than that of the Frame-Stewart algorithm. Due to the varying sizes of the disks on any peg in the starting configuration, it might not be easy to derive a mathematical formula for the creation of an intermediate sub-tower. If a guide is not used, it can be too computationally complex to test every possible set of moves. Disks may need to be repeatedly moved between the four pegs before a condition arises in which it is possible

to create an intermediate sub-tower. Configurations such as those found in Fig. 2 might not be encountered during the regular execution of the Frame-Stewart algorithm. Therefore, unless modifications are introduced to the algorithm, the Frame-Stewart algorithm might not be the best to solve random configurations.
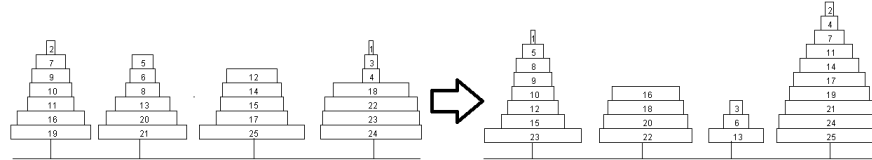

Fig. 2. Start and end configurations of an unsolved random instance

4.2 The Classic 4-peg Problem

The classic 4-peg problem was used to test the same two programs that were tested on the random configurations. For this problem, the programs were tested using an input of 30 disks. The program that used a formula was able to solve the problem using the optimum possible number of moves, 1025 [7]. The one that had no guide ran out of memory.

Since the program that used Rohl's and Gedeon's guide was able to solve the 30-disk problem, it was tested on the classic problem using every number of disks between one and thirty. When these tests were run, the partitioning error bounds of +2 and -2 were removed. For each of the thirty instances, the program was able to solve the problem using the optimum possible number of moves, as proven by [7] and [8].

After these tests were performed, the classic 4-peg problem was used to test the same six partition estimates that were examined using the ASP problem in Table 2. Each estimate was tested on every number of disks between one and thirty. As opposed to the prior tests, these tests did not allow any error bound when calculating the partition number. Although three of the estimates failed to solve every single ASP Competition configuration, all six estimates generated optimal solutions for every instance of the classic problem. Korf has verified the minimum possible number of moves for any algorithm using inputs of up to 30 disks. By using the Frame-Stewart algorithm with any of the six partition estimates on the regular 4-peg Hanoi problem, a solution is obtained that is the optimal solution of any possible algorithm. Tabling and testing every partition number can be too computationally complex, but Rohl's and Gedeon's formula provides a good guide.

5   Alternative Approaches

Many authors have tried to solve the 4-peg and k-peg problems using their own algorithms. Although most of the approaches are not used in this paper, it is important to note their contributions. Following is a subset of the algorithms:

5.0.1   Rohl's and Gedeon's Algorithm. Rohl and Gedeon created recursive algorithms for both the 4-peg and the k-peg problems, using a form of Stewart's algorithm. This paper will focus on the algorithm for four pegs, wherein each recursive call builds a single sub-tower. If the current ordering of the pegs is (1, 2, 3, 4), the 4-peg algorithm is called to create a sub-tower with ordering (1, 4, 3, 2), the 3-peg algorithm is called on (1, 2, 3) to move the remaining pegs, and the 4-peg algorithm is called on (4, 3, 2, 1) to move the sub-tower to the destination peg [13]. This algorithm provided the partitioning guide utilized by BPSolver's modified program.

5.0.2   Lu's Algorithm. Lu produced an iterative approach for the 4-peg problem. Lu's algorithm shows a correlation between the disk moves and binary numbers. It inserts a number of logical fake disks at pre-determined locations in order to map the disk moves to the binary numbers [9].

5.0.3   Sarkar's Algorithm. Sarkar used a recursive algorithm for the k-peg problem that is similar to Frame's algorithm. The algorithm deterministically decides how to distribute the topmost disks over intermediate pegs, using summation functions, before distributing any disks. It then recursively calls itself to distribute the disks. Sarkar posed the *serialization conjecture*, stating that an optimal algorithm will distribute disks onto pegs such that disks on each peg have consecutive sizes, and the size of the top disk on a peg is consecutive with the size of the bottom disk on the next peg. If this conjecture is valid, then Sarkar's algorithm is optimal [14]. Although this distribution method is efficient, this paper does not use it, because it does not easily extend to the configuration problem.

5.0.4   Wang's, Liu's, Yue's, Shao's, and Lu's Algorithm. Another iterative algorithm for the 4-peg puzzle involves arranging the disk numbers into an upper-triangular array. Based on this array, there is a "cross-correlation" between the solution of the problem represented by location (i, j) and the problems represented by locations $(i, j+1)$, and $(i+1, j+1)$. This correlation decreases the number of necessary calculations [22]. This is another efficient algorithm, although there is no clear way to extend it to the configuration problem.

6   Summary

For configurations directly based on the Frame-Stewart algorithm, Rohl's and Gedeon's partitioning estimate of $N - \lfloor \frac{\sqrt{8*N+1}-1}{2} \rfloor$ appears to be an optimal

guide, even if it requires error bounds. Although all of the estimates in Table 2 generate optimal solutions for the regular 4-peg puzzle for up to 30 disks without using error bounds, their correctness does not seem to extend to the configuration problem. Perhaps the nature of the configuration problem changes the optimal partition location.

Testing BPSolver's program on random configurations further clarifies the issue. It appears that there are some configurations for which creating intermediate sub-towers may not be optimal. Unlike the problem as studied by Frame and Stewart, it is not clear where to place the intermediate sub-towers. The non-determinism involved can cause the computation problem to be too computationally complex. However, BPSolver's program with Rohl's and Gedeon's estimates runs quickly, and uses tabling to reduce the number of calculations. Therefore, it is a good guide for a dynamic programming solution for modified forms of the 4-peg Tower of Hanoi problem where non-determinism is required.

The B-Prolog program demonstrates the importance of tabling for declarative description of dynamic programming solutions. Like use of pattern databases in state-space search and conflict-driven clause learning and memoization in SAT solvers, tabling is a great technique for avoiding repeated exploration of the same states during search.

Acknowledgements

References

1. Chu, I-Ping, and Richard Johnsonbaugh. The Four-Peg Tower of Hanoi Puzzle. *ACM SIGCSE Bulletin* Vol. 23, No. 3 (1991), 2-4.
2. Dunkel, Otto. Editorial Note Concerning Advanced Problem 3918. *The American Mathematical Monthly* Vol. 48, No. 3 (1941), 219.
3. Er, M. C. A Note on the Optimality of a Reve Algorithm. *The Computer Journal* Vol. 34, No. 6 (1991), 513.
4. Frame, J. S. Solution for Advanced Problem 3918. *The American Mathematical Monthly* Vol. 48, No. 3 (1941), 216-217.
5. Kaykobad, M, Rahman, S. T.-U., Bakhtiar, R.-A., and A. A. K. Majumdar. A Recursive Algorithm for the Multi-Peg Tower of Hanoi Problem. *International Journal of Computer Mathematics* Vol. 57, No. 1-2 (1995), 67-73.
6. Klavžar, Sandi, and Uroš Milutinović. Simple Explicit Formulas for the Frame-Stewart's Numbers. *Annals of Combinatorics* Vol. 6, No. 2 (2002), 157-167.
7. Korf, Richard E., and Ariel Felner. Recent Progress in Heuristic Search: A Case Study of the Four-Peg Towers of Hanoi Problem. *Proceedings of the 20th International Joint Conference on Artificial Intelligence* (2007), 2324-2329.
8. Korf, Richard E., Zhang, Weixiong, Thayer, Ignacio, and Heath Hohwald. Frontier Search. *Journal of the ACM* Vol. 52, No. 5 (2005), 715-748.
9. Lu, Xue-Miao. An Iterative Solution for the 4-peg Towers of Hanoi. *The Computer Journal* Vol. 32, No. 2 (1989), 187-189.

10. Lunnon, W. F. The Reve's Puzzle. *The Computer Journal* Vol. 29, No. 5 (1986), 478.

11. Majumdar, A. A. K. Generalized Multi-peg Tower of Hanoi Problem. *Journal of the Australian Mathematical Society, Series B-Applied Mathematics* Vol. 38, No. 2 (1996), 201-208.

12. Rand, Michael. On the Frame-Stewart Algorithm for the Tower of Hanoi. Technical Report, Boston College (2009). Available via `https://www2.bc.edu/~grigsbyj/Rand_Final.pdf`.

13. Rohl, J. S., and T. D. Gedeon. The Reve's Puzzle. *The Computer Journal* Vol. 29, No. 2 (1986), 187-188.

14. Sarkar, U. K. On the Design of a Constructive Algorithm to Solve the Multi-peg Towers of Hanoi Problem. *Theoretical Computer Science* Vol. 237, No. 1-2 (2000), 407-421.

15. Stewart, B. M. Solution for Advanced Problem 3918. *The American Mathematical Monthly* Vol. 48, No. 3 (1941), 217-219.

16. Stockmeyer, P. K. The Tower of Hanoi: A Bibliography. Version 2.2 (2005). Available via `http://www.cs.wm.edu/~pkstoc/biblio2.pdf`.

17. Stockmeyer, Paul K. Variations on the Four-Post Tower of Hanoi Puzzle. *Proceedings of the 25th Southeastern International Conference on Combinatorics, Graph Theory, and Computing. Congressus Numerantium* Vol. 102 (1994), 3-12.

18. Third ASP Competition Detailed Scoring Regulations. ASP Competition 2011 Organizing Committee. Available via `https://www.mat.unical.it/aspcomp2011/files/scoringdetails.pdf`.

19. Truszczynski, Miroslaw, Smith, Shaden, and Alex Westlund. Tower of Hanoi: Problem Description. ASP Competition 2011. Available via `https://www.mat.unical.it/aspcomp2011/FinalProblemDescriptions/HanoiTower`.

20. van de Liefvoort, A. An Iterative Algorithm for the Reve's Puzzle. *The Computer Journal* Vol. 35, No. 1 (1992), 91-92.

21. van de Liefvoort, Appie. An Iterative Solution to the Four-Peg Tower of Hanoi Problem. *Proceedings of the 1990 ACM Annual Conference on Cooperation* (1990), 70-75.

22. Wang, Jun, Liu, Junpeng, Yue, Guoying, Shao, Liangshan, and Sukui Lu. A Nonrecursive Algorithm for 4-Peg Hanoi Tower. *2007 International Conference on Intelligent Systems and Knowledge Engineering* (2007).

23. Zhang, Andrew. A Stratification of the Hanoi Graph for 4 Pegs. Technical Report, Columbia University (2008). Available via `http://www.math.columbia.edu/~jason/THreu.Zhang.pdf`.

24. Zhou, Neng-Fa. The language features and architecture of B-Prolog. *Theory and Practice of Logic Programming, Special Issue on Prolog Systems* Vol.12, No. 1-2 (2012), 189-218.

25. Zhou, Neng-Fa, Sato, Taisuke, and Yi-Dong Shen. Linear tabling strategies and optimizations. *Theory and Practice of Logic Programming* Vol. 8, No. 1 (2008), 81-109.

26. Zhou, Neng-Fa, and Christian Theil Have. Efficient Tabling of Structured Data with Enhanced Hash-Consing. *ICLP*, 2012.

27. Zhou, Neng-Fa, Dovier, Agostino, and Yuanlin Zhang. BPSolver's Solutions to the Third ASP Competition Problems. *ALP Newsletter* (June 2011). Available via `http://www.cs.nmsu.edu/ALP/wp-content/uploads/2011/06/Bprolog.pdf`.

28. Zhou, Neng-Fa, Fruhman, Jonathan, and Ligon Liu. Program for Solving the 4-peg Tower of Hanoi Problem. ASP Competition 2011. Available via `http://www.probp.com/asp11/hanoi.pl`.